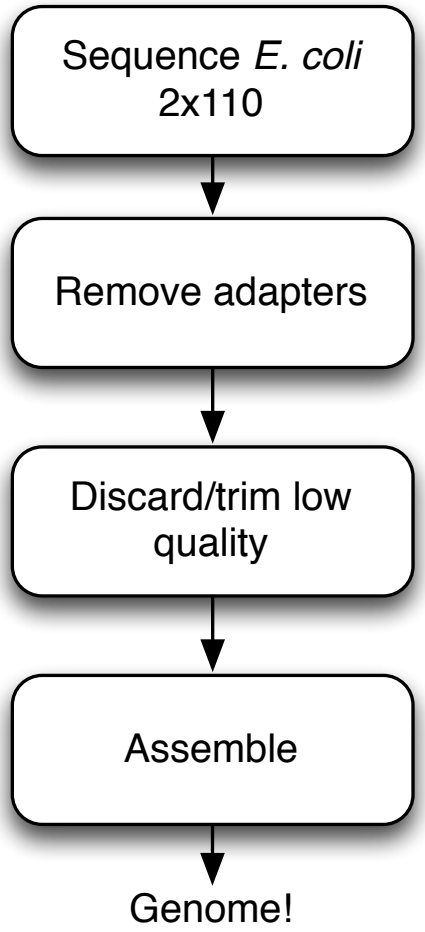
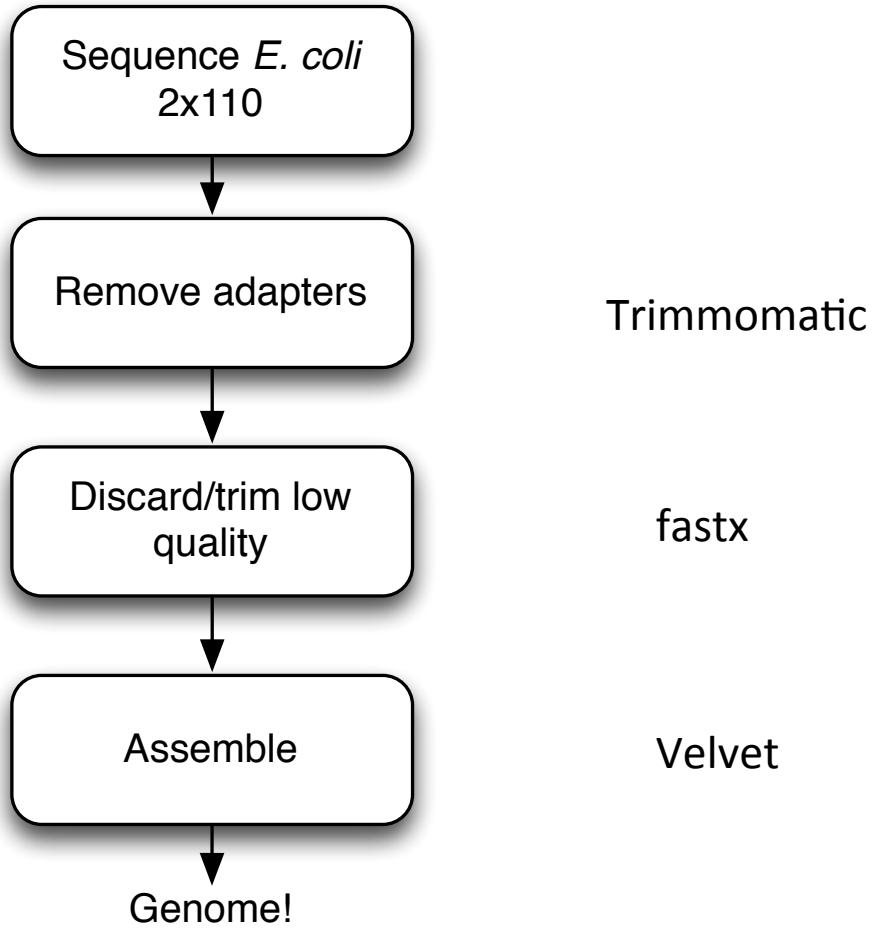
Pipelines!

11/11/13

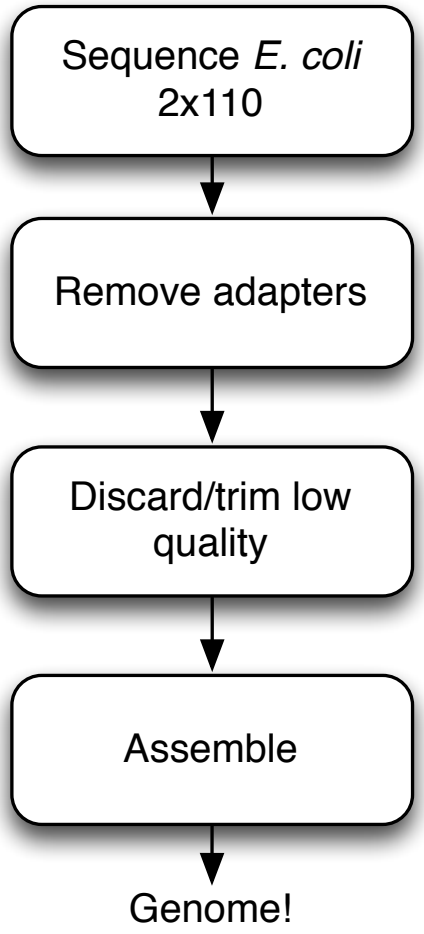
A pipeline view of the world



Each computational step is one or more commands



The breakdown into steps is dictated by input/output…

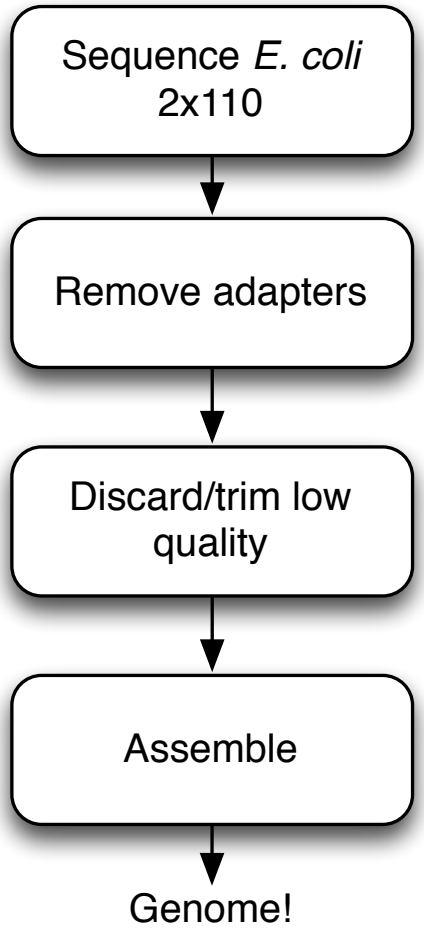


In: reads; out: reads

In: reads; out: reads

In: reads; out: contigs

The breakdown into steps is driven by input/output and "concept"

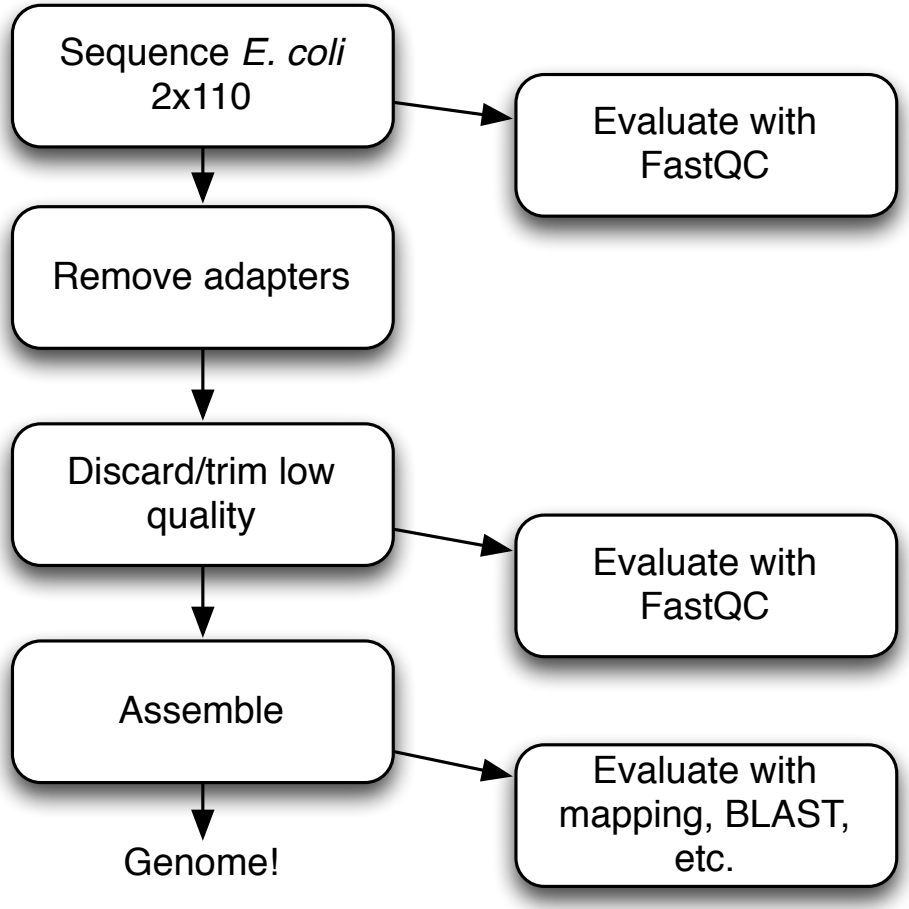


In: reads; out: reads.
Trimmomatic OR scythe OR …

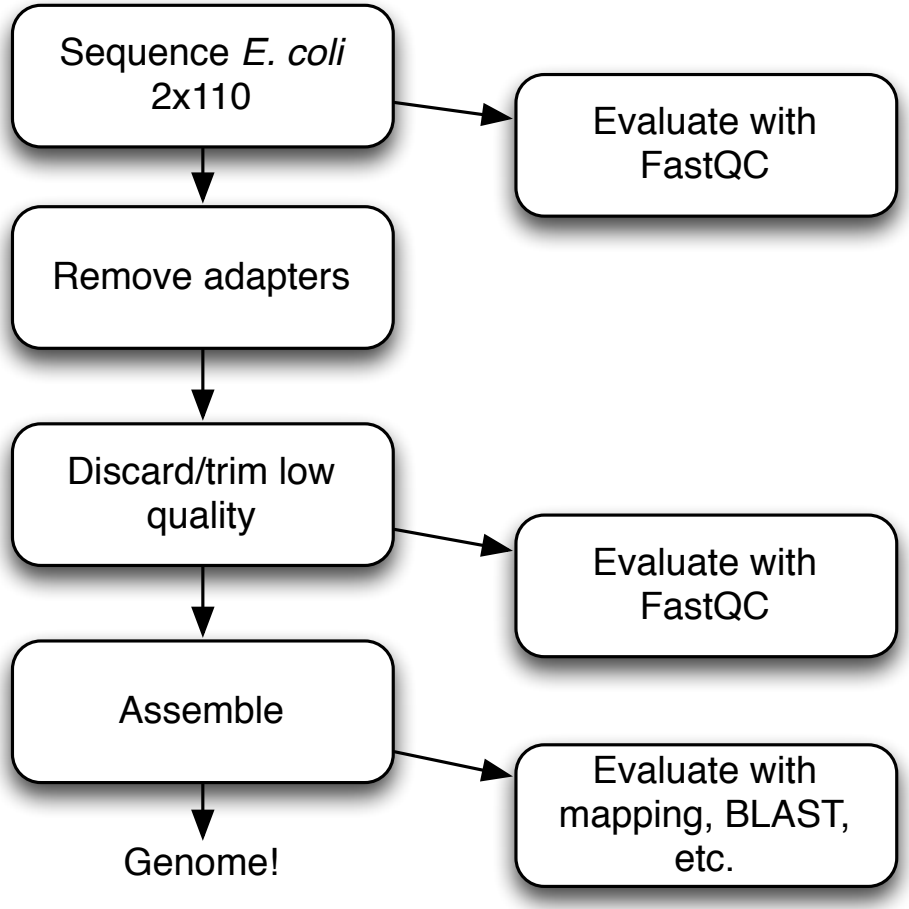
In: reads; out: reads.
FASTX OR sickle OR ConDeTri OR …

In: reads; out: contigs
Velvet OR SGA OR …

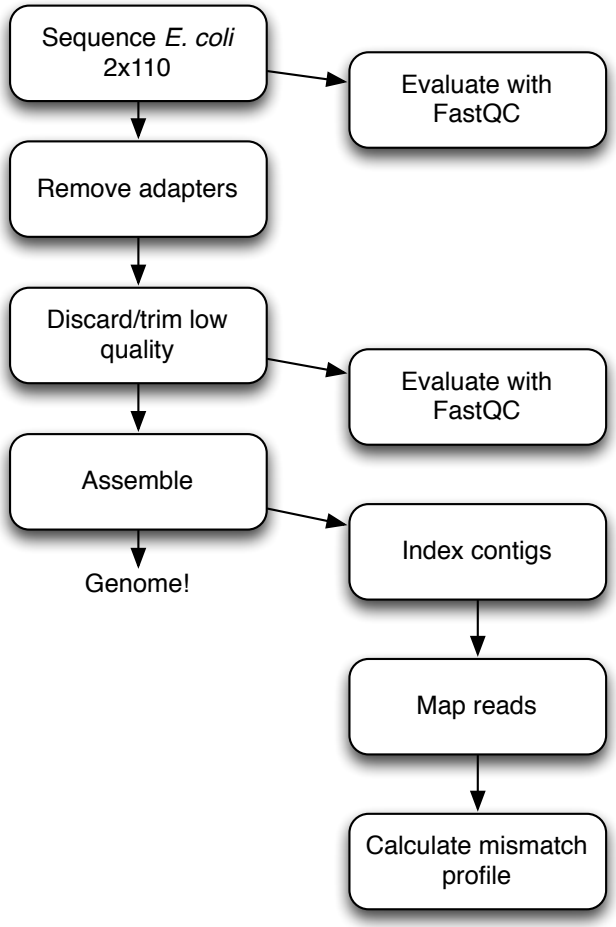
Generally, I don't include diagnostic steps as part of the main "flow".



Generally, I don't include diagnostic steps as part of the main "flow".



…but there isn't exactly a standard :)



What *is* a pipeline, anyway?

- Conceptually: series of data in/data out steps.
- Practically: series of commands that load data, process it, and save it back to disk.
  - This is generally true in bioinformatics
  - You can also have programs that do multiple steps, which involves less disk "traffic"
- Actually: a bunch of UNIX commands.

“Shell scripting”

- The shell (bash, csh, etc) is specialized for exactly this: running commands.
- Shell “scripting” is putting together a series of commands – “scripting actions” to be run.
- Scripting vs programming – fuzzy line.
  - Scripting generally involves less complex organization.
  - Scripting typically done w/in single file

Writing a shell script:

It's just a series of shell commands, in a file.

```
# trim adapters
… Trimmomatic …

# shuffle reads together
Interleave.py …

# Trim bad reads
fastx_trimmer

# Run velvet
velveth...
velvetg…
```

trim-and-assemble.sh

Back to pipelines

- *Automated* pipelines are good things.
  - Encode each and every step in a script;
  - Provide all the details, incl parameters;

- Explicit: each command is present.
- Reusable: can easily tweak a parameter, re-run & re-evaluate.
- Communicable: you can give to lab mate, PI, etc.
- Minimizes confusion as to what you actually did :)
- Automated: start & walk away from long-running pipelines.

Why pipelines?

- Automation:
  - Convenience
  - Reuse
  - **Reproducibility**

Pipelines encode *knowledge* in an explicit & executable computational representation.

Reproducibility

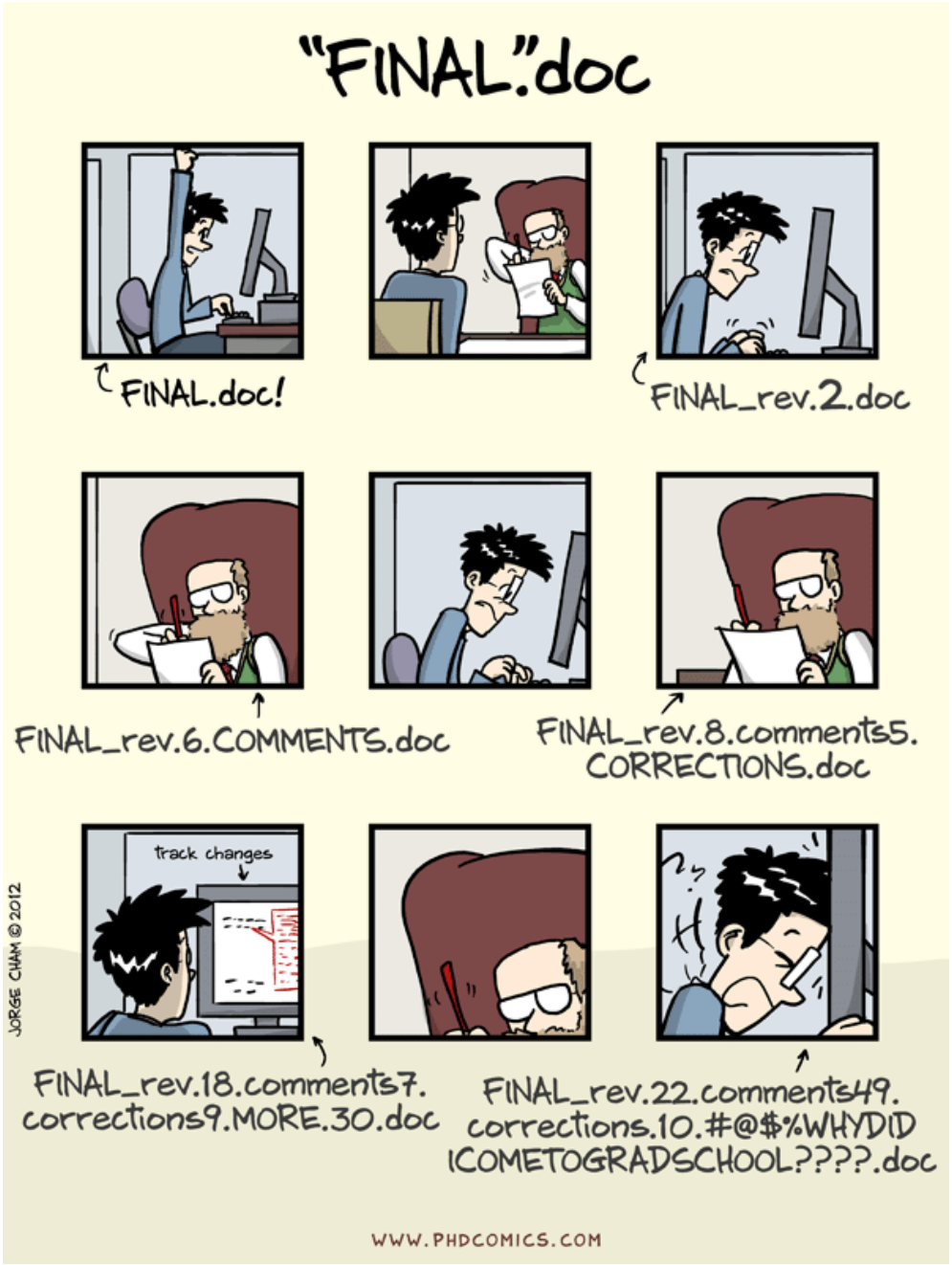
- *Most groups can't reproduce their own results, 6 months later.*
- *Other* groups don't even have a chance.

- Limits:
  - Reusability
  - Bug finding/tracking/fixing

Both *convenience* and *correctness.*

Some nonobvious corollaries

- Each processing step from the raw data onwards is interesting; so you need to provide close-to-raw data.
- Making the figures is part of the pipeline; but Excel cannot be automated.
- Keeping track of what exact *version* of the pipeline script you used to generate the results now becomes a problem…



http://www.phdcomics.com/comics/archive.php?comicid=1531

This is what *version control* is about.

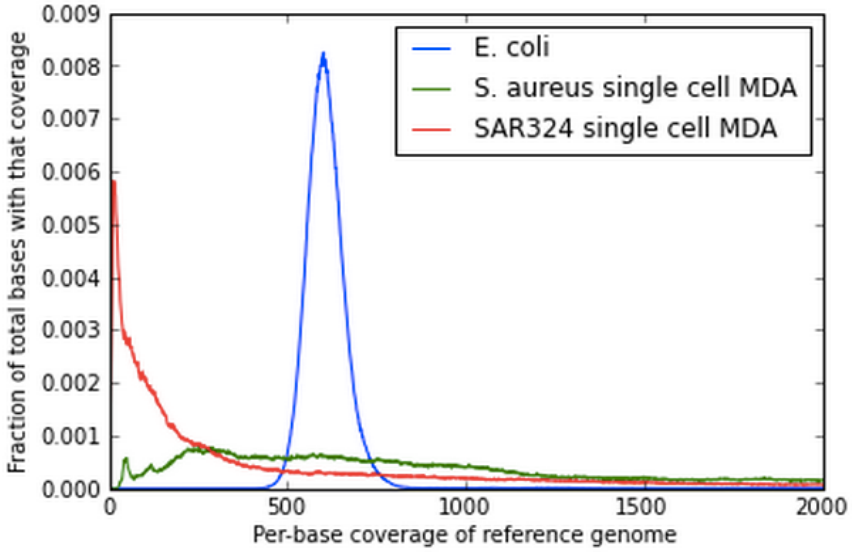
- Version control gives you can explicit way to track, mark, and annotate changes to collections of files.
- (Git is one such system.)
- In combination with Web sites like github.com, you can:
  - View changes and files online
  - Download specific marked versions of files

An actual pipeline

- The results in our digital normalization paper are about 80% automated.
  - Raw data
  - Single command to go from raw data to fully processed data.
  - Single IPython Notebook to go from raw data to figures.
  - (Super special) single command to go from figures + paper source to submission PDF.
  - Figures & text are tied to a specific *version* of our pipeline => 100% reproducible.

IPython Notebook

In [16]:
```
plot(ecoli_cov[:,0], ecoli_cov[:,1])
plot(staph_cov[:,0], staph_cov[:,1])
plot(sar_cov[:,0], sar_cov[:,1])
xlabel("Per-base coverage of reference genome")
ylabel("Fraction of total bases with that coverage")
legend(["E. coli", "S. aureus single cell MDA", "SAR324 single cell MDA"])
axis(xmax=2000)
savefig('/tmp/diginorm-coverage2-raw.pdf')
```



In [17]:
```
ecoli_kcov = numpy.loadtxt(datadir + 'ecoli.keep.rawreads.map.gz.cov')
```

This afternoon

- Shell introduction
  - Commands, files, directories
  - A little bit of scripting: examples
- A quick tour of github
  - Forking, cloning, editing, pushing back
- Assembly & data exploration

Tips & tricks

- Develop a systematic naming scheme for files => easier to investigate results.
- Work with a small data set first & develop the pipeline; then, once it's working, apply to full data set.
- Put in friendly "echo" commands.
- Advanced: use *loops* and *wildcards* to write generic processing steps.